

České vysoké učení technické v Praze  
Fakulta stavební

155UZPD: Semestrální projekt

# **MMDOS - Síťové analýzy PID**

Únor 2019

Petra Mariana Millarová, Michael Kala

# 1 Cíl projektu

Cílem semestrálního projektu bylo vytvoření konzolové aplikace MMDOS, jež provádí síťové analýzy nad daty PID za využití pgRouting, extenze PostGISu. Uživatel má možnost zadat výchozí a cílovou adresu, aplikace vyhledá nejbližší zastávky hromadné dopravy, provede síťovou analýzu a vrátí nejkratší cestu. Uživateli se poté zobrazí seznam zastávek a linek, které by měl po cestě využít.

## 2 Data

- Adresní místa RÚIAN Hlavního města Prahy, staženo z Nahlížení do KN (dokumentace: [http://vdp.cuzk.cz/vymenny\\_format/csv/ad-csv-struktura.pdf](http://vdp.cuzk.cz/vymenny_format/csv/ad-csv-struktura.pdf))
- Síť tras linek Pražské integrované dopravy, staženo z portálu Opendata Hlavního města Prahy (dokumentace: [http://www.geoportalpraha.cz/cs/fulltext\\_geoportal?id=6F576389-385E-4E38-831A-8DE6EFB52C3A#.XErV8stKiV4](http://www.geoportalpraha.cz/cs/fulltext_geoportal?id=6F576389-385E-4E38-831A-8DE6EFB52C3A#.XErV8stKiV4))
- Zastávky PID - jednotlivé označníky, staženo z portálu Opendata Hlavního města Prahy (dokumentace: [http://www.geoportalpraha.cz/cs/fulltext\\_geoportal?id=63EF19FE-C2FB-4FC2-8C2D-EEBB72C6B81A#.XErXJ8tKiV4](http://www.geoportalpraha.cz/cs/fulltext_geoportal?id=63EF19FE-C2FB-4FC2-8C2D-EEBB72C6B81A#.XErXJ8tKiV4))

## 3 Zpracování dat

### 3.1 Adresní místa RUIAN

Data byla stažena ve formátu `csv`. Následně pomocí jazyka AWK byl v shellu soubor upraven, aby obsahoval pouze potřebná data (resp. byl vytvořen nový soubor):

```
1 awk -F \; ' { print $11";"$13";"$14";"$15";"$17";"$18 } ' 20181231
   _OB.554782_ADR.csv > ruian_adr.csv
```

Ponechány byly sloupce ulice, číslo domovní, číslo orientační, číslo orientační znak (a, b..) a souřadnice x, y v S-JTSK.

Dále byla v databázi na geo102 vytvořena a naplněna tabulka `adr` (včetně geometrie) pomocí dávkového souboru `adr.sql` puštěného pomocí příkazu:

```
1 psql -h geo102.fsv.cvut.cz -d pgis_uzpd -U uzpd18_a -f adr.sql
```

Obsah dávkového souboru:

```
1 CREATE TABLE adr(
2   gid serial NOT NULL,
3   ulice VARCHAR(50) ,
4   c_domovni INTEGER,
5   c_orientacni INTEGER,
6   co_znak VARCHAR(2) ,
7   y REAL,
8   x REAL,
9   geom geometry);
10
11 \copy adr(ulice , c_domovni, c_orientacni , co_znak , y, x) FROM '../ruian_adr
   .csv' DELIMITER ';' CSV HEADER encoding 'windows-1250';
12
13 UPDATE adr SET geom = ST_GeomFromText('POINT(-' || y || ' -' || x || ')', 5514);
14
15 DELETE FROM adr WHERE geom IS NULL;
```

Aby byly názvy ulic importovány správně s diakritikou, bylo třeba nastavit kódování na uvedenou hodnotu `windows-1250`. Zároveň byla pro potřeby projektu z dat vymazána adresní místa bez geometrie.

### 3.2 Data PID

Data zastávek a tras linek PID byla stažena ve formátu `shp`. Pro import dat do databáze byl využit nástroj PostGISu `shp2pgsql`, jež konvertuje soubory ve formátu `shp` do databázových tabulek. Toto bylo vykonáno pomocí příkazu (v shellu):

```
1 shp2pgsql -s 5514 -D -I DOP.PID-TRASY-TSL.shp | psql -h geo102.fsv.cvut.cz
   -d pgis_uzpd -U uzpd18_a
```

A přímo v databázi byl změněn název vytvořené tabulky:

```
1 ALTER TABLE dop_pid_trasy_ts_l RENAME TO trasy;
```

Pro potřeby našeho projektu nebyly nutné údaje o nočních linkách, proto byly tyto řádky smazány (soubor `mazani_nocnich.sql`):

```
1 DELETE FROM trasy
2 WHERE (l_metro_n IS NULL
```

```

3 OR l_tram_n IS NOT NULL
4 OR l_bus_n IS NOT NULL
5 OR l_lan_n IS NOT NULL
6 OR l_vlak_n IS NOT NULL
7 OR l_lod_n IS NOT NULL
8 AND l_metro IS NULL
9 AND l_tram IS NULL
10 AND l_bus IS NULL
11 AND l_lan IS NULL
12 AND l_vlak IS NULL
13 AND l_lod IS NULL);

```

Analogicky se postupovalo u zastávek:

```

1 shp2pgsql -s 5514 -D -I DOP.PID_ZASTAVKY.TS.B.shp | psql -h geo102.fsv.cvut.
  cz -d pgis-uzpd -U uzpd18_a

```

```

1 ALTER TABLE dop_pid_zastavky_ts_b RENAME TO zastavky;

```

U zastávek byly taktéž smazány řádky obsahující pouze údaje o nočních linkách:

```

1 DELETE FROM zastavky WHERE zast_denno = 2;

```

Vzhledem k prapodivnosti sloupce `zast_id`, kde se ukázalo, že více zastávek s různými názvy a různým umístěním mají stejné ID, se autoři rozhodli použít sloupec `zast_uzel` jako identifikátor zastávky, protože byla hodnota stejná pro všechny položky se stejným názvem zastávky. Sloupec bylo třeba přetypovat na typ `INTEGER` následujícím příkazem:

```

1 ALTER TABLE zastavky
2 ALTER COLUMN zast_uzel_ TYPE INTEGER
3 USING CAST(zast_uzel_ AS INTEGER);

```

## 4 Topologie

### 4.1 Hrany

Vzhledem ke komplikacím zmiňovaným v předchozí kapitole se autoři rozhodli vytvořit topologii bez použití funkce z extenze `pgRouting` k tomu určené - `pgr_createTopology`. Nejprve byly do tabulky `trasy` přidány sloupce `source` a `target` a vytvořen prostorový index (soubor `topo.sql`):

```

1 ALTER TABLE trasy ADD COLUMN "source" integer;
2 ALTER TABLE trasy ADD COLUMN "target" integer;
3 CREATE INDEX ON trasy USING gist(geom);

```

Následně byl vytvořen skript v Pythonu, který pomocí prostorového dotazu k začátku i konci každé linie v tabulce `trasy` vybral nejbližší bod z tabulky `zastavky` a vrátil ID jeho uzlu, které pak bylo dosazeno do sloupce `source`, případně `target`. Vzhledem k tomu, že geometrickým typem linií v tabulce `trasy` byl `MultiLineString` a funkce `ST_StartPoint()` resp. `ST_EndPoint()` jako argument bere pouze typ `LineString`, bylo třeba mezi těmito typy provést konverzi.

Celý skript (soubor *MMTopology.py*):

```
1 import psycopg2
2
3
4 # connect to database
5 conn = psycopg2.connect(host="geo102.fsv.cvut.cz",
6                          database="pgis_uzpd",
7                          user="uzpd18_a",
8                          password="a_uzpd18")
9
10 cur = conn.cursor()
11 cur_s = conn.cursor()
12 cur_e = conn.cursor()
13
14 # Select starting points
15 cur_s.execute("SELECT DISTINCT ON(t.gid) t.gid, z.zast_uzel_ FROM trasy t,
16               zastavky z WHERE ST_DWithin(ST_StartPoint(ST_LineMerge(t.geom)), z.geom,
17               500) AND t.zast_id_od = z.zast_id")
18
19 # Select end points
20 cur_e.execute("SELECT DISTINCT ON(t.gid) t.gid, z.zast_uzel_ FROM trasy t,
21               zastavky z WHERE ST_DWithin(ST_EndPoint(ST_LineMerge(t.geom)), z.geom,
22               500) AND t.zast_id_ka = z.zast_id")
23
24 sp = cur_s.fetchone()
25 ep = cur_e.fetchone()
26 i = 0
27
28 # loop through starting and ending points, update source and target
29 while sp is not None or ep is not None:
30     if sp is not None:
31         cur.execute("UPDATE trasy SET source = {} WHERE gid = {}".format(sp[1],
32                                   sp[0]))
33     if ep is not None:
34         cur.execute("UPDATE trasy SET target = {} WHERE gid = {}".format(ep[1],
35                                   ep[0]))
36     sp = cur_s.fetchone()
37     ep = cur_e.fetchone()
38
39 conn.commit()
40
41 cur_s.close()
42 cur_e.close()
43 cur.close()
44
45 if conn is not None:
46     conn.close()
```

Počet řádků v tabulce trasy se pohybuje kolem 85 000, skript běžel několik desítek minut.

## 4.2 Uzly

Následně byla vytvořena tabulka uzlů pomocí funkce *pgr\_createVerticesTable*. Sloupec *the\_geom* byl přejmenován na *geom* (soubor *uzly.sql*):

```
1 SELECT pgr_createVerticesTable('trasy','geom','source','target');
2 ALTER TABLE trasy_vertices_pgr rename column the_geom to geom;
```

## 5 Konzolová aplikace

### 5.1 Popis výpočtu

V jazyce Python byl vytvořen skript (soubor *MMDOS.py*), který na vstup uživatele provádí síťovou analýzu za využití Dijkstrova algoritmu implementovaného v *pgRouting* jako funkce *pgr\_dijkstra*. Postup výpočtu je následující:

1. Zjištění ID nejbližších zastávek od zadaných adres, aplikace umožňuje uživateli zadat následující kombinace údajů o adresách:

- ulice, č.p./č.o.
- ulice, č.p.
- ulice, č.o.
- ulice

kdy na základě zadaných údajů je vyhledána adresa jim vyhovující, případně první z adres jim vyhovující.

2. Výpočet nejkratší cesty pomocí Dijkstrova algoritmu.

3. Výpis zastávek a linek respektující nejmenší počet přestupů.

- V tabulce *trasy* jsou pro každý úsek vyjmenovány všechny linky, které na daném úseku poskytují přepravu. Program prochází vytvořenou trasu a vybírá linky s nejmenším počtem přestupů následovně:
  - (a) Průchod všech linek projíždějících první zastávkou (resp. úsekem mezi první a druhou zastávkou), výpočet dosahu každé linky (tedy až o kolik zastávek se lze posunout danou linkou).
  - (b) Přiřazení čísla linky s nejdelším dosahem (příp. první z linek, které mají shodný nejdelší dosah) k daným zastávkám.
    - Vzhledem k přiřazování čísel linek k počátečním bodům úseků, se číslo linky nepřidá k poslední zastávce, do které vybraná linka dosahuje. Toto je využito v dalším kroku.
  - (c) Opakování kroku (a) a (b) od první zastávky s nepřirazeným číslem linky.

## 5.2 Vytvořené SQL funkce

Pro potřeby konzolové aplikace byly vytvořeny následující SQL funkce (soubor `funkce.sql`):

- Funkce pro zjištění názvu zastávky podle zadaného ID.

```
1 CREATE OR REPLACE FUNCTION FindStationName(id INTEGER)
2 RETURNS VARCHAR AS $name$
3 declare
4     name varchar;
5 BEGIN
6     SELECT z.zast_nazev INTO name FROM zastavky z
7     WHERE z.zast_uzel_ = id LIMIT 1;
8     RETURN name;
9 END;
10 $name$ LANGUAGE plpgsql;
```

- Set funkcí pro zjištění ID uzlu zastávky, která je nejbližší zadané adrese. První tři funkce jsou uzpůsobeny na zadání názvu ulice a kombinace čísla orientačního, popisného a nebo obou. Poslední z funkcí umožňuje uživateli zadat jen název ulice, přičemž funkce vrací i adresu, ke které bylo hledání nejbližší zastávky vztaženo. U prvních dvou funkcí není třeba vracet celou adresu, jelikož čísla popisná a orientační jsou dle zákona unikátní v rámci ulice (popisné v rámci části obce).

```
1 CREATE OR REPLACE FUNCTION FindVertexID(cd INTEGER, co INTEGER, u
2     VARCHAR)
3 RETURNS INTEGER AS $id$
4 declare
5     id integer;
6 BEGIN
7     SELECT v.id INTO id FROM adr a, trasy_vertices_pgr v
8     WHERE a.c_domovni = cd
9     AND a.c_orientacni = co
10    AND a.ulice = u
11    ORDER BY (a.geom)<->(v.geom) asc limit 1;
12    RETURN id;
13 END;
14 $id$ LANGUAGE plpgsql;
15
16 CREATE OR REPLACE FUNCTION FindVertexIDcd(cd INTEGER, u VARCHAR)
17 RETURNS INTEGER AS $id$
18 declare
19     id integer;
20 BEGIN
21     SELECT v.id INTO id FROM adr a, trasy_vertices_pgr v
22     WHERE a.c_domovni = cd
23     AND a.ulice = u
24    ORDER BY (a.geom)<->(v.geom) asc limit 1;
25    RETURN id;
26 END;
27 $id$ LANGUAGE plpgsql;
28
29 CREATE OR REPLACE FUNCTION FindVertexIDori(co INTEGER, u VARCHAR)
30 RETURNS INTEGER AS $id$
31 declare
32     id integer;
```



```

32 BEGIN
33     SELECT v.id INTO id FROM adr a, trasy_vertices_pgr v
34     WHERE a.c_orientacni = co
35     AND a.ulice = u
36     ORDER BY (a.geom)<-(v.geom) asc limit 1;
37     RETURN id;
38 END;
39 $id$ LANGUAGE plpgsql;
40
41 CREATE OR REPLACE FUNCTION FindVertexIDst(u VARCHAR)
42 RETURNS TABLE(
43     id BIGINT,
44     ul VARCHAR,
45     cp INTEGER,
46     co INTEGER
47 ) AS $$
48 BEGIN
49     RETURN QUERY SELECT v.id, a.ulice, a.c_domovni, a.c_orientacni FROM
50         adr a, trasy_vertices_pgr v
51     WHERE a.ulice = u
52     ORDER BY (a.geom)<-(v.geom) asc limit 1;
53 END;
54 $$
55 LANGUAGE plpgsql;

```

### 5.3 Dijkstraův algoritmus

Pro výpis nejkratší cesty byl použit následující příkaz:

```

1 SELECT z.zast_nazev, trasy.l_metro, trasy.l_tram, trasy.l_bus, trasy.l_lan,
   trasy.l_vlak, trasy.l_lod, node
2 FROM pgr_dijkstra('SELECT gid AS id, source, target, CAST(shape_leng AS
   REAL) AS cost FROM trasy', id_from, id_to)
3 JOIN (
4     SELECT DISTINCT(zastavky.zast_uzel_), zastavky.zast_nazev FROM zastavky
5 ) AS z ON node = z.zast_uzel_
6 LEFT JOIN trasy ON edge = trasy.gid
7 ORDER BY seq;

```

kde `id_from`, `id_to` jsou ID počáteční a koncové zastávky.

K tabulce výsledků z výpočtu funkce `pgr_dijkstra` byla přes sloupec `zast_uzel_` připojena tabulka `zastavky`, ze které byl vypsán název zastávky. Připojena byla též tabulka `trasy` pro zobrazení linek, které daným místem projíždějí. Protože poslední uzel nemá žádnou navazující hranu, ale musí být také vypsán, byl použit `LEFT JOIN`, který zachovává všechny údaje z původní tabulky. Nakonec byl výstup seřazen podle atributu `seq`, jež udává pořadí zastávek v nalezené trase.

## 5.4 Ukázka aplikace

Aplikace lze spustit spuštěním skriptu `MMDOS.py` případně v OS Linux pomocí stand-alone spustitelného souboru `MMDOS` (v přílohách).

```
Connecting to database pgis_uzpd on geo102.fsv.cvut.cz...
Connected!

-----
MMDOS ©
-----
Created by: Maru & Michael 2019
-----

Start with typing from where you want to travel:
Street: Evropská
House number: 612
House orientation number:

Where do you want to go?
Street: Thákurova
House number:
House orientation number: 7

Your route from Evropská 612/79 to Thákurova 2077/7:
-----
      Stop | Line
-----
      Sídliště Červený Vrch | Get on: 20
      Bořislavka | 20
      Na Pískách | 20
      Hadovka | 20
      Thákurova | Transfer to: 108
      Studentský dům | Get off here!
-----

Again? (Y/N): n
Exiting..
```

## 6 Závěr

### 6.1 Zhodnocení

Byla vytvořena konzolová aplikace pro vyhledávání spojů PID v závislosti na zadané adrese počátečního a cílového místa. Aplikace respektuje nejmenší počet přestupů.

Při zpracování dat se vyskytlo několik problémů, zejména u dat poskytovaných na serveru Opendata, kde docházelo k případům, že několik zastávek mělo stejné "unikátní ID", přestože se nejmenovaly stejně a ani se nenacházely na stejném dopravním uzlu. Stejná ID byla použita v datové sadě s linkami PID u ID počátečních a koncových bodů jednotlivých úseků linek. Toto vyústilo v problém, kdy docházelo k chybnému přiřazení zastávek k příslušným úsekům linek. Tento problém byl vyřešen pomocí prostorového dotazování na nejbližší zastávky k daným bodům.

Dalším problémem byla přítomnost záznamů linek s nočními spoji. To způsobovalo chybné vyhledávání nejkratších tras. Toto bylo vyřešeno vymazáním těchto údajů z tabulky.

### 6.2 Náměty na vylepšení

- Ocenění tras např. podle typu dopravy - současný model hledá nejkratší trasu bez ohledu na přestupy
- Přidání možnosti nočního cestování
- Přidání jízdních řádů

## 7 Přílohy

- Import adres do databáze - `adr.sql`
- Mazání údajů o nočních linkách - `mazani_nocnich.sql`
- Topologie tras (přidání sloupců) - `topo.sql`
- Skript pro topologii tras - `MMTopology.py`
- Vytvoření tabulky s uzly - `uzly.sql`
- Skript MMDOS - `MMDOS.py`
- Vytvořené SQL funkce - `funkce.sql`
- Stand-alone aplikace - MMDOS

## 8 Zdroje

- Nahlížení do KN, aplikace ČÚZK - <https://nahlizeniidokn.cuzk.cz/>
- Portál pro Otevřená data hlavního města Prahy - <http://opendata.praha.eu/>
- Upload csv tabulky do databáze - <http://www.postgresqltutorial.com/import-csv-file-into-postgresql-table/>
- Tvorba topologie pomocí pgRouting - [http://docs.pgrouting.org/latest/en/pgr\\_createTopology.html](http://docs.pgrouting.org/latest/en/pgr_createTopology.html)
- Tvorba nové tabulky uzlů - [http://docs.pgrouting.org/latest/en/pgr\\_createVerticesTable.html](http://docs.pgrouting.org/latest/en/pgr_createVerticesTable.html)
- Funkce pro vyhledání nejkratší trasy - [http://docs.pgrouting.org/latest/en/pgr\\_dijkstra.html](http://docs.pgrouting.org/latest/en/pgr_dijkstra.html)